Optimization and Hardware Implementation of Image Watermarking for Low Cost Applications

Konstantinos Pexaras[1], Christos Tsiourakis[2], Irene G. Karybali[1] and Emmanouil Kalligeros[1]

[1]Dept. of Information and Communication Systems Engineering, University of the Aegean, 83200 Karlovassi, Samos, Greece
Emails: icsd09119@icsd.aegean.gr, {karybali, kalliger}@aegean.gr

[2]Cisco Systems, Amsterdam, The Netherlands, Email: ctsioura@cisco.com

***Abstract*—In many resource/ power constrained applications (e.g., in wireless-network nodes) there is a need for low cost hardware implementations of digital image watermarking techniques. However, to end up with such an implementation, a proper selection of a watermarking technique, based on performance and cost criteria, is not enough. For that reason, in this paper we introduce computation optimizations of the implemented algorithm to keep the integer portion of arithmetic operations at optimal size, and, hence, arithmetic units as small as possible. Additionally, the proposed architecture reutilizes those units in different computation steps, to further reduce implementation cost. The proposed design compares favorably with the already existing implementations, in terms of area, power and performance.**

I. Introduction

The prevalence of wireless networks has made the long-term need for communications security more imperative. Apart from the sensitive nature of the transmitted data (e.g., surveillance cameras), wireless devices are vulnerable to physical attacks, while the wireless technologies used for their communications are susceptible to interference and interception [1], [2]. Since, in many applications, the critical data are images, image security is a major concern for such networks.

Two of the required security features are copyright protection and authentication. A malicious adversary may duplicate a node's data for profit [3], while the source of critical or security data must be verifiable. Digital watermarking has been successfully used for copyright protection and authentication of images or other kinds of data. Concerning digital images, many watermarking techniques have been proposed in the literature (e.g., see [4]). In many cases though, the cost of wireless nodes has to be kept low, which means that their resources (mainly, processing power and memory) are very limited [1]. Therefore, a new low-cost environment emerges, in which the implementation, in software, of many of the proposed digital image watermarking algorithms is not possible.

In such cases, a hardware implementation is preferable. A number of hardware implementations of image watermarking techniques have been presented in the literature (refer, e.g., to [5], [6]). However, for wireless-network applications, a hardware-implemented watermarking scheme should fulfill two contradictory requirements: high robustness to attacks, due to the critical nature of the data, and small implementation cost. The latter is necessary because of the stringent power consumption constraints (i.e., battery life) of wireless nodes.

To address both of these demands at the same time, in this paper we develop a robust, low-cost watermark embedder. First, a watermarking technique is properly selected, based on performance and cost criteria. In terms of cost, what matters is that the necessary arithmetic operations require, mainly, "cheap" arithmetic units. Moreover, since implementation cost comes primarily from these units, they should be as small as possible. To this end, we introduce computation modifications that maintain the integer portion of operations at optimal size. Finally, a hardware architecture is proposed that reutilizes arithmetic units in different computation steps, to keep the overall cost as small as possible.

II. Watermark Embedding

Since a hardware-implemented watermarking technique for low cost applications should be robust and have a small footprint, we opt for spatial domain techniques based on perceptual masking. We avoid the cost of the direct and inverse transformations involved in frequency domain embedding, and, using an appropriate perceptual mask, we ensure the watermark robustness. By employing perceptual masks, which take into account the properties of the human visual system (HVS), the watermark energy can be properly adapted, achieving robustness and invisibility at the same time. In this work, we adopt the NVF (noise visibility function)-based mask presented in [7] for watermark embedding, which is robust to JPEG compression and other attacks [4]. JPEG compression can cause image blocking and blurring, which are also common artifacts introduced by wireless channel losses.

If x is the cover image, w the watermark and M_{NVF} the NVF-based mask, the watermarked image is

$$y = x + \alpha M_{NVF} \odot w, \tag{1}$$

where $\odot$ stands for point-wise multiplication. The watermark is a Gaussian distributed pseudorandom pattern of equal size and uncorrelated with the cover image. It has zero mean, while its strength (i.e., its standard deviation) is denoted by α, which means that w itself has variance equal to 1. The watermark for a camera can be generated off-line and stored in a memory.

The NVF identifies textured and edge regions, where the watermark can be amplified. For the determination of the optimal NVF, the watermark is considered as noise, and the classical MAP image denoising approach is used [7]. The perceptual mask derived, assuming a non-stationary Gaussian model, is defined as

$$M_{NVF}(i,j) = 1 - \frac{1}{1+\sigma_x^2(i,j)}, \tag{2}$$

where $\sigma_x^2(i,j)$ is the local variance of the cover image in a neighborhood centered on pixel (i,j), $1 \le i \le M$, $1 \le j \le N$ and $M \times N$ are the image dimensions.

978-1-5386-1911-7/17/$31.00 ©2017 IEEE

A. Optimization for Low Cost Hardware Implementation

In this sub-section, we introduce computation modifications that maintain the integer portion of operations at optimal size. Since, after watermarking, the values of gray-scale images can be ≥ 256 or <0, the optimal size of their (signed) integer portion is 10 bits. Additionally, 10 bits are used for their fractional part to have sufficient precision. Below, we determine the peak value that each involved quantity can reach and we propose proper modifications, so as the involved computations fit in the 10.10 (20-bit wide) signed fixed-point representation.

The mask is computed using the unbiased sample variance:

$$\sigma_x^2(i,j) = \frac{1}{mn-1} \sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} [x(i+k,j+l) - \mu(i,j)]^2, \quad (3)$$

where

$$\mu(i,j) = \frac{1}{mn} \sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} x(i+k,j+l) \quad (4)$$

is the local mean value computed in a $m \times n$ neighborhood and $\lfloor \cdot \rfloor$ is round to the integer towards minus infinity.

The sums' result in (4) is upper bounded by $255mn$ (total white neighborhood), which needs $\lceil \log_2(255mn) \rceil$ bits for its representation ($\lceil \cdot \rceil$ is round to the integer towards plus infinity). We rearrange the operations in (4) as shown below:

$$\mu(i,j) = \sum_{i=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{j=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{mn} \cdot x(i+k,j+l). \quad (5)$$

We first multiply the pixels' values with $\frac{1}{mn}$ and then compute their sum, which now takes values in the interval $[0, 255]$. So, only 9 bits are needed for the representation of its integer part.

Continuing with (3), the difference of the image and the local mean values is in the range $(-255, 255)$ (the endpoint values can never occur), and the sums' result, which takes values in $[0, 255^2 mn)$, needs $\lceil \log_2(255^2 mn) \rceil$ bits for the representation of its integer part. Instead of $\sigma_x^2(i,j)$, we compute $\frac{\sigma_x^2(i,j)}{256}$ which is equal to

$$\sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \frac{[x(i+k,j+l) - \mu(i,j)]}{mn-1} \cdot \frac{[x(i+k,j+l) - \mu(i,j)]}{256}. \quad (6)$$

We divide $[x(i+k,j+l) - \mu(i,j)]$ with $(mn-1)$ and with 256, we multiply the results and compute their sum. Since we use a 3×3 neighborhood in our implementation, $\frac{1}{mn-1}$ translates to 3 right shifts. Similarly, division by 256 corresponds to 8 right shifts. The range of values of $\frac{\sigma_x^2(i,j)}{256}$ is $[0, \frac{255^2}{(mn-1)256} \cdot mn)$, and for a 3×3 neighborhood, 10 bits are needed for the representation of its integer part.

Finally, the mask is computed by the following equation:

$$M_{NVF}(i,j) = 1 - \frac{\frac{1}{256}}{\frac{1}{256} + \frac{\sigma_x^2(i,j)}{256}}. \quad (7)$$

Concerning the desired image quality, it can be derived by adjusting the watermark strength, i.e. α, in eq. (1). The peak signal-to-noise ratio ($PSNR$) is used for this purpose:

$$PSNR = 10 \log_{10} \frac{\max(x)^2}{\|y - x\|^2} \quad (8)$$

where $\max(x) = 255$ and $\| \cdot \|$ is the Euclidean norm. Let $u \equiv M_{NVF} \odot w$ be the masked watermark. Then, from (8):

$$\alpha = \frac{A}{\sqrt{\|u\|^2}}, \qquad A = \frac{255}{\sqrt{10^{\frac{PSNR}{10}}}}. \quad (9)$$

A is smaller than 9 when the $PSNR$ ranges from 30 to 45 dB. Higher values of $PSNR$ correspond to better image quality. Our implementation gives the option to select the quality level, by using a lookup table with different values of A.

The computation of α involves the computation of

$$\|u\|^2 = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} u^2(i,j), \quad (10)$$

where $u^2(i,j) = M_{NVF}^2(i,j) \cdot w^2(i,j)$ is upper bounded by $w^2(i,j)$, since M_{NVF} takes values from 0 to 1. Since w is a Gaussian distributed pseudorandom pattern with zero mean and variance equal to 1, $\frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} w^2(i,j)$ equals the variance of w, i.e. equals 1 [8]. This means that the sums' result in (10), i.e. $\|u\|^2 MN$, is upper bounded by MN, which needs $\lceil \log_2 MN \rceil$ bits for its representation (we remind that $M \times N$ are the image dimensions).

To reduce the number of bits required for the above representation, we modified the computation of $\|u\|^2$ as follows:

$$\|u\|^2 = \frac{1}{256} \sum_{q=1}^{J} \left[\left[\left(\sum_{p=P_{q-1}+1}^{P_q} \mathbf{u}^2(p) \right) / \left(\frac{M}{16} \right) \right] / \left(\frac{N}{16} \right) \right], \quad (11)$$

where vector $\mathbf{u}$ contains the elements of the masked watermark taken column-wise, $P_0 = 0$ and $P_q : \sum_{p=P_{q-1}+1}^{P_q} \mathbf{u}^2(p) \leq 462$. First, the elements of $\mathbf{u}$ are added until their sum reaches value 462. We remind that u is upper bounded by w, which takes values in $(\mu - 6\sigma, \mu + 6\sigma)$, i.e. $(-6, 6)$ in our case, with probability that approaches 1. In order to ensure that $\sum_{p=P_{q-1}+1}^{P_q} \mathbf{u}^2(p)$ will not exceed 511, we set a threshold equal to 462 $(511 - 7^2)$. This threshold was selected to cover the extreme case that w takes values out of $(-6, 6)$. Then, this sum is divided first with $M/16$ and then with $N/16$. Note that by using the 10.10 signed fixed-point representation, the image dimensions can be up to $2^{13} \times 2^{13}$. The above process is repeated for all elements of u. The total sum, i.e. $256 \cdot \|u\|^2$, is upper bounded by 256 and needs 10 bits for its signed, integer part representation. Its square root is then computed and $\alpha = (A \times 16)/\sqrt{256 \cdot \|u\|^2}$. Since $\|u\|^2$ is upper bounded by 1, α in (9) can take a bit higher values than A.

Finally, the watermarked image y in (1) takes values within the desired representation range, since $\alpha M_{NVF} \odot w$ is upper bounded by $\alpha \cdot w$, which takes values in $(-6\alpha, 6\alpha)$.

III. Hardware Implementation

The architectural diagram of the implemented watermark-embedding module is shown in Fig. 1. The computation blocks are drawn as squares. To keep area and power consumption low, we decided to process only one pixel at a time. In this way, only one of the blocks of Fig. 1 is used every instant, which allows us to share arithmetic units among the blocks.

In our implementation, we assumed that the captured images are stored as raw, gray-scale data in a RAM, from which

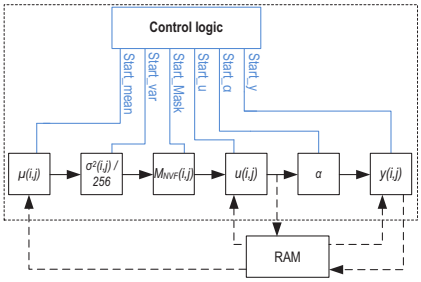

Fig. 1. Architecture of the implemented module

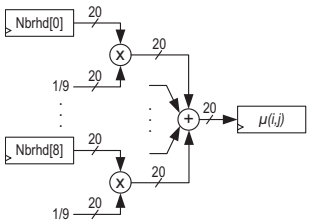

Fig. 2. μ-block datapath - Parallel implementation

they are retrieved. We note that, to keep Fig. 1 simple, we do not show the implemented RAM interface but only the necessary RAM accesses. This is why the corresponding lines are dashed. We should also mention that the u-block reads the watermark from the RAM and writes back the $u(i,j)$ values. The latter is done because all $u(i,j)$ values are needed first for the calculation of the watermark strength α, and then for computing the watermarked image y.

A. Implementation of the μ-block

In Fig. 2, the parallel implementation of the μ-block datapath is shown. Actually, we have developed two different versions of the μ- and the $\sigma^2/256$-blocks: a serial, targeting very low cost, and a parallel for improved performance. In the parallel version of the μ-block, nine multipliers are employed for multiplying every pixel of a neighborhood (Nbrhd[0]-Nbrhd[8]) with $1/9$ (Eq. 5) in a single clock cycle. The resulting products are then added by a tree of eight adders. On the contrary, in the serial version, only one multiplier and one adder are used, but the mean value's computation is performed in nine successive clock cycles. We note that after being read from the RAM, every pixel's value is augmented with twelve 0's (10 on the right and 2 on the left side) so as to be transformed to the desired 10.10 fixed-point format. Also, any neighborhood pixels outside the image boundaries are considered as 0's. As far as the multiplications are concerned, from their 40-bit results, we keep the middle 20.

B. Implementation of the $\sigma^2/256$-block

The datapath of the parallel version of the $\sigma^2/256$-block is shown in Fig. 3. First, the difference of the neighborhood's mean value $\mu(i,j)$ from every neighborhood pixel is calculated (by 2's complement addition) and the results are stored in registers (diff[0]-diff[8]). This latching is done for clock-speed optimization purposes. Then, the multiply-add structure of the μ-block (9 multipliers and 8 adders) is re-used to compute the sum of products of Equation (6). This structure is fed with the

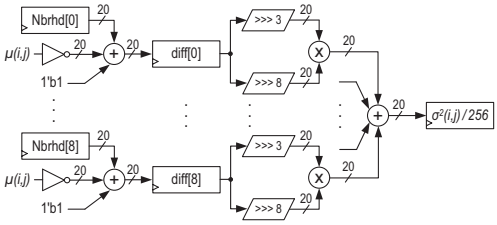

Fig. 3. $\sigma^2/256$-block datapath - Parallel implementation

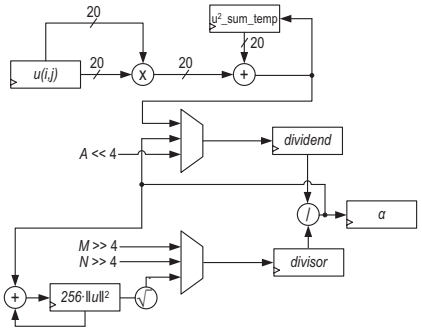

Fig. 4. α-block datapath ($<<$, $>>$ denote shifts)

arithmetically right-shifted versions by 3 (divide by 8) and by 8 (divide by 256) of the *diff* registers. The serial version of the $\sigma^2/256$-block needs, instead of nine, just one additional adder (but extra clock cycles) for performing the subtractions. In this case too, the multiplier and the adder of the serial μ-block are re-used by the serial $\sigma^2/256$-block.

C. Implementation of the M_{NVF}-, u- and y-blocks

We discuss these blocks very briefly. The u-block performs just a multiplication, while the y-block a multiplication and an addition. The M_{NVF}-block is a bit more complex because it requires an addition, a division and a subtraction (i.e., addition). Since division is an expensive operation, we pipelined the divider in order to optimize throughput (see Section IV).

D. Implementation of the α-block

The block calculating the watermark strength α is the most complex of our implementation and is shown in Fig. 4. Each $u(i,j)$ value is first multiplied by itself so as to be squared and is then added to the value of a register that keeps the temporary sum of the squared $u(i,j)$'s (u^2*_sum_temp*). When the value of that register becomes equal or greater than 462, as explained in Section II-A, it is sent through the upper multiplexer of Fig. 4 to the divider, and is divided first by $M/16$ and then by $N/16$. The final quotient is added to a register that keeps the partial $256 \cdot \|u\|^2$ value. The same process is repeated for all $M \times N$ $u(i,j)$'s and, when completed, the total $256 \cdot \|u\|^2$ value is first square-rooted and then fed to the divider (through the lower multiplexer) as a divisor of $A \times 16$. Note that we have pre-computed 16 different values of A, which correspond to 16 different $PSNRs$ ($30 - 45dB$). These values are stored in a small lookup table of our system.

As far as the arithmetic units are concerned, the square root gets a 20-bit input and outputs a 10-bit value, which corresponds to a 5.5 fixed-point integer in our implementation.

TABLE I
DESIGN CHARACTERISTICS AND SYNTHESIS RESULTS

	Cyclone IV E		ASIC	
	Serial Impl.	Parallel Impl.	Serial Impl.	Parallel Impl.
Area [FPGA: # Logic Elements (% utilization), ASIC: mm^2]	4399 (3.8%)	4582 (4.0%)	1.52	2.89
Total 1-bit Registers (FPGA only)	1969	1968	-	-
Clock frequency (MHz)	88.69	79.84	181.82	166.7
ASIC Power Dissipation (mW)	-	-	4.69	4.23
# Integer arith. units (add, mul, div, sqrt)	7, 1, 1, 1	22, 9, 1, 1	7, 1, 1, 1	22, 9, 1, 1
# Pipeline stages (div, sqrt)	12, 4	11, 2	5, 2	6, 2

For that reason, its outputs are augmented with 10 0's (5 on the left and 5 on the right side) to be transformed to the proper internal representation. As for the divider, to yield results with 10 fractional bits, the dividend should be padded with 10 0's on the right side ($\equiv \times 1024$). Hence, the corresponding register should be 30-bit wide. The quotient needs 30 bits as well, from which we keep the 20 least significant that correspond to the required 10.10 fixed-point result. Finally note that, similar to the divider, the square root unit is also pipelined (Section IV).

IV. EXPERIMENTAL RESULTS AND COMPARISONS

Both the serial and the parallel version of the described watermark embedder were implemented in Verilog and mapped to the Cyclone IV E EP4CE115F29C7 FPGA device, using Intel's (ex Altera) Quartus Prime tool, and to an industrial 90 nm standard-cell library (ASIC), using a commercial synthesis tool. The Cyclone IV FPGA is the main chip of Terasic's DE2-115 board, which we used, apart from post-synthesis simulations, for verifying our designs. Thus, we also implemented an SRAM interface for accessing the board's SRAM, during our hands-on experiments. All results, along with some important design information are presented in Table I. It can be seen that the total number of arithmetic units has been kept very low: 10 (33) overall in the serial (parallel) implementation, 7 (22) of which are adders. Observe that we have limited the usage of the costly division and square root units to just 1 instance each, which are shared among different blocks. The same holds for the multipliers, the total number of which depends on the serial or parallel implementation of the μ- and $\sigma^2/256$- blocks (any other multipliers needed are from this set). In our designs, we have not restricted only the adders' volume, due to their small size (20 bits). As for the pipelined units, different stage volumes were chosen, depending on performance. Note that the results of Table I also include the SRAM-interface's overhead.

An important advantage of the proposed embedder is that its parallel ASIC version can support video. Specifically, we have averaged the clock cycles needed for various 640×480 gray-scale images (small discrepancies exist, since the number of divisions during α's calculation is not fixed), and the achieved throughput is 31.9 frames/sec (the serial design achieves 18 frames/sec). Note that in our architecture, all SRAM read/write cycles, apart from those in the y's calculation, are overlapped with computation cycles. Frame rates can be significantly improved if we pipeline the computation blocks of Fig. 1, which we intend to do as future work (line buffers will be also used for streaming the frames' pixels to the embedder).

Since the existing literature designs are mapped to various technologies or FPGA devices, a direct comparison with them is quite difficult. Due to space limitations, we provide here the characteristics' range of invisible, robust watermarking implementations like ours. Thus, for ASIC designs in the literature, area is between 1.46 and $16.2mm^2$, power is between 0.3 (a low-power implementation) and $62.78mW$, and clock frequency lies between 50 and $280MHz$, while for FPGAs, clock freq. ranges between 52.4 and $82.26MHz$. It is obvious that the proposed embedder, in terms of area and power, lies at the lower end of the corresponding ranges. This is due to the few, small-sized arithmetic units it incorporates. Its clock speeds are quite high too. Note also, that video capability is offered, which is not a characteristic of the compared designs.

The performance of the proposed embedder was also compared to the original algorithm's software implementation. We computed the mean squared error (MSE) among various watermarked images derived from the board and those generated in Matlab with floating-point double-precision representation. The mean MSE, for 640×480 images, is from 0.6829 for 30 dB to 0.0215 for 45 dB, which is fairly small, as expected considering the utilized fixed-point representation.

V. CONCLUSIONS

We have presented an image watermarking hardware implementation that suits well the needs of low cost applications. The proposed design offers low area/power due to the introduced computation optimizations that keep arithmetic units small, and the reuse of the latter in different computation steps. Also, performance is high enough to allow video support, for the parallel, ASIC version of our design, for 640×480 frames.

REFERENCES

[1] Q. M. Ashraf and M. H. Habaebi, "Autonomic schemes for threat mitigation in Internet of Things," *Journal of Network and Computer Applications*, vol. 49, pp. 112–127, March 2015.

[2] D. O. Gonçalves and D. G. Costa, "A survey of image security in wireless sensor networks," *Journal of Imaging*, vol. 1, no. 1, pp. 4–30, 2015.

[3] R. Xiao, X. Sun, and Y. Yang, "Copyright protection in wireless sensor networks by watermarking," in *Proc. of Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, Aug. 2008, pp. 7–10.

[4] I. G. Karybali and K. Berberidis, "Efficient spatial image watermarking via new perceptual masking and blind detection schemes," *IEEE Trans. on Information Forensics and Security*, vol. 1, pp. 256–274, June 2006.

[5] F. M. Idris, *Advanced Techniques in Multimedia Watermarking: Image, Video and Audio Applications*. IGI Global, 2010, ch. 17, Hardware Implementations of Image/Video Watermarking Algorithms, pp. 425–454.

[6] H. K. Maity and S. P. Maity, "FPGA implementation of reversible watermarking in digital images using reversible contrast mapping," *Journal of Systems and Software*, vol. 96, pp. 93–104, Oct. 2014.

[7] S. Voloshynovskiy, A. Herrigel, N. Baumgaertner, and T. Pun, "A stochastic approach to content adaptive digital image watermarking," in *Proc. of the 3rd Int. Workshop of Information Hiding*, Sep. 1999, pp. 211–236.

[8] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*. McGraw Hill, 4^{th} edition, 2002.